From Domain Model to High-Mix Low-Volume Production Line Schedules

Jacques Verriet
TNO-ESI
Eindhoven, the Netherlands
Email: jacques.verriet@tno.nl

Bram van der Sanden
TNO-ESI
Eindhoven, the Netherlands
Email: bram.vandersanden@tno.nl

Twan Basten
Electronic Systems group
Eindhoven University of Technology
Eindhoven, the Netherlands
Email: a.a.basten@tue.nl

***Abstract*— To maximize performance in modern production environments, there is a trend toward integrating multiple systems from different vendors into production lines. Scheduling work in such lines requires careful coordination of the machines' operational states. Achieving optimal schedules involves considering the duration of operations, travel, and sequence-dependent setup – factors that are critical in a high-mix low-volume (HMLV) setting due to the variability in job requirements. We present a methodology for scheduling HMLV production lines that explicitly considers these time constraints. The approach employs a modular domain model to describe the production line and the diverse set of operations it must perform. Domain model specifications are translated into a constraint graph, which enables efficient computation of a shortest feasible production line schedule. To demonstrate our methodology, we apply it to a production scenario inspired by the professional printing domain. The results illustrate the applicability of the approach to production environments in which flexibility and responsiveness are essential.**

I. Introduction

Traditionally, high-tech equipment is designed to achieve maximum performance, typically measured in throughput and output quality, in isolation. In practice, however, such machines are increasingly deployed as part of a larger context, often a cyber-physical production system that integrates equipment from multiple vendors. Examples include production printers operating within a printshop and lithography equipment within a wafer fab. The effective performance of a system in such a context depends partly on its standalone performance, but primarily on the performance of the production system into which it is integrated.

In this paper, we focus on *production lines*, fully automated production systems with limited internal buffer capacity. To optimize the performance of a production line, the line's machines cannot be considered in isolation. Each machine must be considered within its operational context, i.e., the line's other machines. Because of the limited buffer capacity, one needs to consider the availability of all equipment when scheduling work. Work arriving too early may lead to collisions, as the incoming materials cannot be stored. Such collisions lead to expensive machine stalls or even downtime. A machine may need to reduce its speed to match the speed of a downstream machine, or it may need to delay its operation to guarantee just-in-time (JIT) delivery of material at the next machine.

A *feasible schedule* ensures that (1) materials arrive at times when the machines on their paths are ready to accept them and (2) materials leave machines directly after the materials' operations on the machines finish. A machine's ability to receive material depends on its state, i.e., its history of operations. Before a machine can accept new material, it must have completed all prior operations. However, it may also need to prepare for the new material. The duration of this preparation depends on the previously handled material and is called *sequence-dependent setup time*. Such setup times may be significantly longer than the operation durations. These setup times are specifically relevant for *high-mix low-volume (HMLV) production systems*, which produce short series of a large variety of products.

This paper describes a methodology to compute shortest feasible production line schedules. The methodology's input is a modular domain model consisting of four domain-specific languages (DSLs). The DSLs describe a production line in terms of (1) materials and their operations, (2) the equipment's machines, (3) jobs and their operations, and (4) sequences of job operations per machine. We compute shortest feasible schedules using constraint graphs [1], which are generated from the domain model. *Constraint graphs* can be seen as a special kind of constraint program, which can be analyzed efficiently. Constraint graphs are particularly suited for scheduling production systems with setup times and JIT constraints [2], [3].

This paper is organized as follows. Related work is presented in Sec. II. Sec. III introduces an illustrative case inspired by the professional printing domain. This case is used in Sec. IV to introduce the domain model and in Sec. V to explain and evaluate the schedule synthesis using constraint graphs. Conclusions are presented in Sec. VI.

II. Related Work

In its general form, scheduling involves two types of decisions: (1) the allocation of operations to available resources and (2) sequencing the operations allocated to the same resource [4]. Sequencing can be further subdivided into (a) determining the order of operations and (b) determining their start and end times. In the context of scheduling production lines, operation allocation is often predetermined; this is, for instance, the case for flow shop and job shop scheduling, which only consider sequencing.

There are many variants of shop scheduling problems [5] including variants with sequence-dependent setup times and

JIT constraints, commonly referred to as blocking and no-wait constraints. Shop scheduling is a notably complex problem, which is typically addressed by mixed integer linear programming (MILP), constraint programming (CP), or (meta)heuristics [6], [7], [8]. Whereas flow shop and job shop scheduling are NP-hard, the scheduling problem considered in this paper can be solved in polynomial time. This is because we assume that both the allocation of operations and their execution order are given, and only the timing of operations must be determined.

Regarding domain modeling approaches for shop scheduling, Eclipse LSAT [9] is closely related to our work. LSAT supports modeling production systems using domain-specific languages (DSLs) and enables timing analysis. Machines are modeled as resources containing peripherals that can execute actions, and jobs are modeled as activities comprising actions and their functional and timing dependencies. In contrast to our approach, LSAT does not support sequence-dependent setup times.

Tsai et al. [10] introduce a formalism for specifying relative release and due dates, which can be used to model sequence-dependent setup times and JIT constraints in production lines. Their approach is based on timing constraint Petri nets, which extend timed Petri nets with minimum and maximum delay constraints between transitions. They use simulation to determine whether a timed Petri net marking is reachable from its initial marking without time constraints. Next, they analyze whether the marking is also reachable when considering the time constraints. If it is, then they synthesize an earliest and a latest schedule. Compared to our methodology, their approach may fail to find a feasible schedule, if one exists, and it lacks an accessible means to specify timing constraint Petri nets.

Like our work, Waqas et al. [2] and Van Pinxten et al. [3] address scheduling with release and due dates. They also use constraint graphs to compute a schedule. Their focus, however, is on re-entrant jobs on a single machine, where the interleaving of the re-entrant operations is optimized, i.e., the order of operations is only partially given. In contrast, we consider multiple scenarios involving multiple machines in which the operation sequences per machine are given. While our methodology supports re-entrant scenarios, it assumes that the interleaving is provided as input.

Where our methodology uses constraint graphs to compute the timing of operations, Van Wanrooij et al. [11] use homogeneous synchronous dataflow graphs (HSDFGs). They extend these with negative operation execution times to specify due dates/JIT constraints, mimicking the way such constraints are captured in constraint graphs. The extended HSDFGs get transformed into $(\max, +)$ linear systems to calculate a shortest schedule. For a practical implementation, Van Wanrooij et al. [11] propose the development of DSLs, something that is a key element of our methodology.

III. Book Binding Example

To illustrate our approach, we use a fictitious book binding scenario inspired by the professional printing domain. The production equipment is shown in Fig. 1, where rectangles represent machines and triangles represent material buffers. The equipment involves two printers, one that prints book blocks and one that prints book covers. The printers retrieve sheets from separate input buffers. The cover printer is connected to a laminator, which laminates printed covers. The block printer feeds a buffer that collects the printed sheets of a single book block before binding. Because this buffer only allows retrieval of its entire contents, it should not store sheets from multiple book blocks simultaneously. Consequently, the block printer may need to pause until the block buffer containing a previous book block has been emptied.

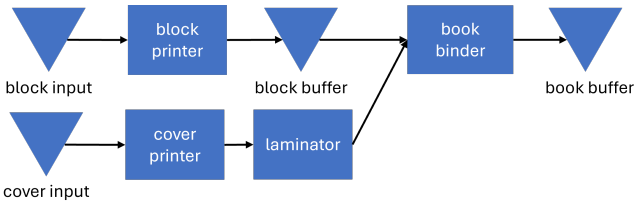

Fig. 1. Book binding equipment.

The block buffer and the laminator are connected to the book binder, which assembles a book from a cover and a book block. The book binder requires that book blocks and covers arrive simultaneously. As there is no buffer between the cover printer, the laminator, and the book binder, printing and lamination of covers may need to wait until the corresponding book block has been retrieved, printed, and gathered.

We assume that the equipment can produce books in two sizes: small and large. Accordingly, the input materials are small and large sheets for both book blocks and covers. All machines can handle both small and large materials. Operation durations depend on material size, with large materials requiring more time than small ones. In addition, all machines incur large costs for switching between small and large materials. These sequence-dependent setups take significantly more time than the operations themselves, which makes it advantageous to process books of the same size consecutively. This may, however, not always be feasible because of delivery constraints.

IV. Domain Model

In this section, we present the domain model to describe production line scheduling scenarios. The domain model involves four DSLs: (1) a Material DSL, (2) an Equipment DSL, (3) a Job DSL, and (4) an Allocation DSL. We explain these DSLs in Sects. IV-A, IV-B, IV-C, and IV-D using the book binding case introduced in Sec. III. The DSLs have a textual syntax defined using Xtext[1] and a graphical frontend called PLOT (Production Line Optimization Tool). Both are explained in more detail in [12].

[1] https://eclipse.dev/Xtext/

```
basic material type sheet

composite material type block composed of sheet

basic material type book
```

Fig. 2. Definition of material types.

```
basic material sheetS type sheet
  dimensions  length = 0.3 m  width = 0.2 m  height = 0.0001 m

composite material blockS type block composed of sheetS
  dimensions  length = 0.3 m  width = 0.2 m

basic material bookS type book
  dimensions  length = 0.3 m  width = 0.2 m
```

Fig. 3. Definition of small material instances.

A. Material DSL

The Material DSL describes the materials handled by a production line and the operations performed on them. The DSL defines material types and instances of these types. In the book binding case, we consider three material types: *sheets*, *book blocks*, and *books*. These material types each have two instances, corresponding to sheets, book blocks, and books of small and large size. Fig. 2 shows the definition of the material types and Fig. 3 the small material instances. Note that the material type book block is a composite material consisting of the inner sheets of a book.

The Material DSL's operations are defined in terms of their input and output materials. The DSL distinguishes five types of operations: *normal*, *storage*, *retrieval*, *assembly*, and *disassembly*. The book binding case involves two retrieval operations (*retrievesheet* and *retrieveblock*), two storage operations (*storesheet* and *storebook*), two normal operations (*print* and *laminate*), and one assembly operation (*bind*). Fig. 4 shows three of these operations.

B. Equipment DSL

The Equipment DSL describes the *nodes*, i.e., machines and buffers, in a production line and the *segments* that connect them.

For each node, the DSL specifies the operations that it can perform and the corresponding durations, which depend on the handled materials. One can also specify the machines' sequence-dependent setup times. These are defined based on the machine's previous operation and the handled materials. Fig. 5 shows the book binding case's equipment. Note that Fig. 5 closely resembles Fig. 1.

Fig. 6 shows the definition of an equipment node with its operations and the corresponding durations and setup

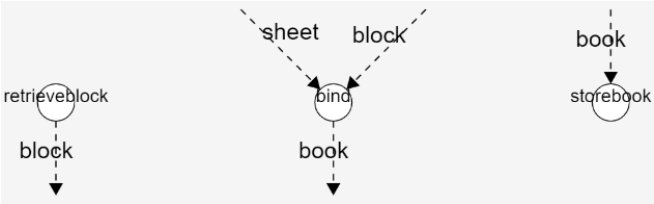

Fig. 4. Operations *retrieveblock*, *bind*, and *storebook*, visualized in PLOT.

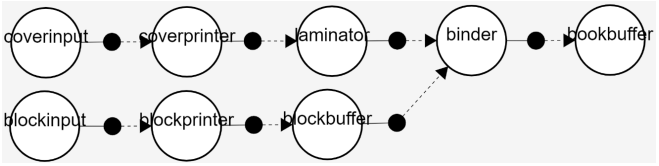

Fig. 5. Equipment topology, in PLOT.

```
buffer node coverinput
    output out type sheet
    operation retrievesheet
        output sheetS duration 2 s
            after retrievesheet output sheetS setup 1 s
            after retrievesheet output sheetL setup 10 s
        output sheetL duration 4 s
            after retrievesheet output sheetS setup 10 s
            after retrievesheet output sheetL setup 1 s
```

Fig. 6. Definition of an equipment node.

times. From the specification, one can see that retrieving large sheets (i.e., *sheetL*) takes more time than retrieving small sheets (i.e., *sheetS*). Fig. 6 also shows that the setup times needed for switching between small and large sheets are much longer than the operation durations.

Fig. 7 shows the specification of an equipment segment. The segment captures the physical transport of material (sheets for this segment). The model assumes that the transport duration is independent of the material instance.

C. Job DSL

The Job DSL describes batches of products that need to be produced, i.e., the workload. Each job concerns one *product* consisting of *parts* that are produced by a sequence of operations. A batch in the book binding case involves multiple books, where one book corresponds to a job. These books have varying thicknesses, i.e., the number of sheets in the book block, and varying sizes, i.e., small or large. As shown in Fig. 8, binding a book requires four parts. The first part involves retrieving book block sheets from the block input buffer, printing them, and storing them in the book block buffer. The second part involves retrieving the printed book block sheets from the book block buffer. The second part can start when the first part has finished. The third part can run in parallel to the first and second part. It involves retrieving a sheet from the cover input buffer and then printing and laminating it. The fourth part follows the second and third part: it binds a cover and a book block into a book, which gets stored in the book output buffer.

D. Allocation DSL

The Allocation DSL specifies the allocation of a batch defined using the Job DSL onto the equipment defined using the Equipment DSL. For each operation in a job, one must

```
segment cover1 type sheet
    source node coverinput output out
    target node coverprinter input in
    duration 1 s
```

Fig. 7. Definition of an equipment segment.

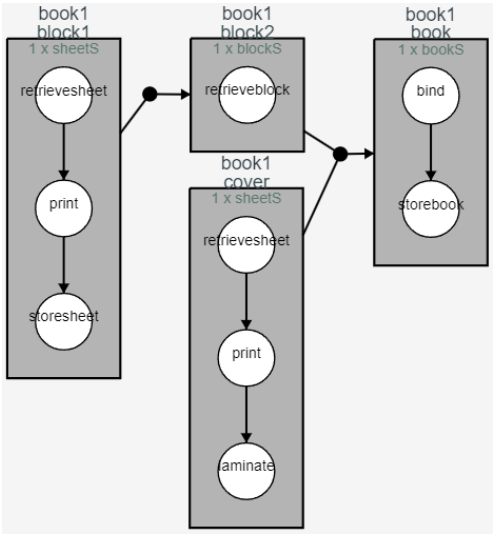

Fig. 8. A book binding job, in PLOT.

```
product book2
    part block1
        blockinput:     retrievesheet
        blockprinter:   print
        blockbuffer:    storesheet

    part block2
        blockbuffer:    retrieveblock

    part cover
        coverinput:     retrievesheet
        coverprinter:   print
        laminator:      laminate

    part book
        binder:         bind
        bookbuffer:     storebook
```

Fig. 9. Allocation of the production of one book.

specify which machine will perform it. Fig. 9 shows the allocation of the production of one book. It shows that the retrieval, printing, and storage of book block sheets are allocated to the block input, block printer, and block buffer, respectively. An allocation needs to be specified for all parts in a batch.

Apart from the allocation of operations to the equipment, the Allocation DSL also specifies the order in which the operations are to be executed: this sequencing is defined by the order of the parts in the Allocation DSL. This sequencing must match the dependencies between the parts as specified in the Job DSL. For instance, one cannot schedule the retrieval of a book block before its sheets have been printed.

Besides the dependencies within a job, the Allocation DSL also specifies the order in which operations of different jobs are executed on the corresponding machines. In other words, all operation ordering is specified in the Allocation DSL. The Allocation DSL allows parts of different products to be interleaved, but this is not feasible in the book binding study as there is no buffer capacity to store multiple book blocks or multiple covers for future handling.

V. Schedule Synthesis

From a production line scheduling scenario specified in the domain model, we can compute a shortest feasible production line schedule. In our setting, a feasible schedule is an assignment of a start and end time to all operations that satisfies all time constraints. Our methodology computes a feasible schedule that minimizes the *makespan*, i.e., the maximum operation end time.

A. Constraint Graph Generation

To compute a shortest feasible schedule, we use a transformation from the domain model to a *constraint graph* [1]. A constraint graph is a directed graph, whose nodes represent events and whose weighted arcs represent time constraints between these events. A time constraint between events e_1 and e_2 is of the form $t_2 \geq t_1 + \Delta$, where t_1 and t_2 represent the times at which events e_1 and e_2 occur and $\Delta \in \mathbb{R}$ represents the minimum time between the events. A non-negative value of Δ represents a (relative) *release date* for event e_2: it must occur at least Δ time after e_1. A negative value of Δ specifies a (relative) *due date* for event e_1: it must occur at most $-\Delta$ time after e_2.

Time constraints for operations, transport, and sequence-dependent setup can all be captured by constraint graphs. For instance, sequential execution of two operations by the same machine can be modeled by a single arc with weight 0 between the end event of the first operation and the start event of the second operation. Moreover, we can represent an operation of (positive) duration Δ by two events, a start event s and an end event e, and two arcs, one from s to e with weight Δ and one from e to s with weight $-\Delta$. JIT arrival of a transport can be modeled in this way as well.

These examples show that constraint graphs may contain cycles. Cycles correspond to events having (transitive) time constraints with themselves. If such a cycle does not have a positive total weight, its time constraints can be satisfied. However, a positive-weight cycle represents constraints that cannot be satisfied.

To generate a constraint graph from a domain model instance, we create a start and an end event for each operation. Fig. 10 shows a constraint graph fragment involving constraints between multiple parts of one product. It shows the start and end events of several operations. The *retrieval* and *arrival* nodes represent starts of operations and the *storage* and *departure* nodes correspond to ends of operations. Fig. 10 also shows transport constraints, i.e., the constraints between ends and starts of consecutive operations. The highlighted *bidirectional* arc captures the combination of a release and a due date constraint with the same absolute weight. This arc captures the transportation and JIT constraints between the end of the lamination of a cover to the start of the book binding. Similarly, there is a JIT constraint between the retrieval of a book block and the start of book binding. This means that a book's cover and book block must arrive simultaneously at the book binder.

Fig. 11 shows another constraint graph fragment. This fragment involves the events corresponding to the retrieval

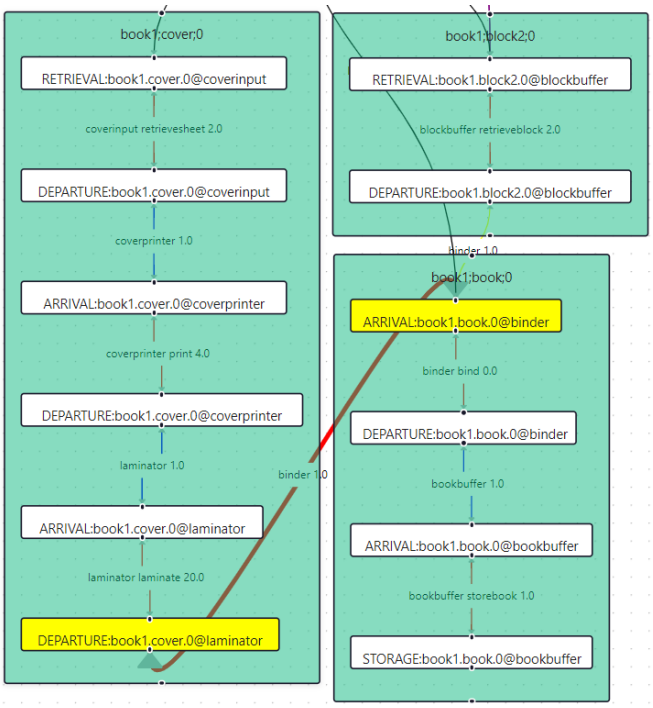

Fig. 10. Constraint graph fragment with JIT constraints, in PLOT.

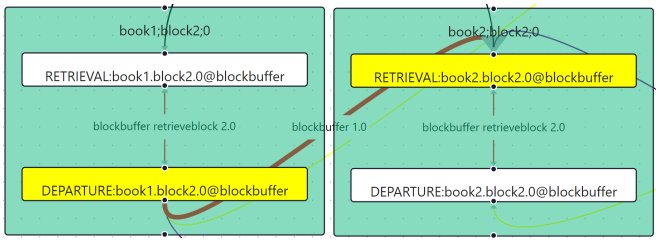

Fig. 11. Constraint graph fragment with setup constraints, in PLOT.

of two consecutive book blocks from the block buffer. There are bidirectional arcs between the starts and the ends of the *retrieveblock* operations representing the duration of the operations. Fig. 11 also contains an arc between the first and second retrieval operation: the highlighted arc represents the setup time of the block buffer between retrieving the first and second book block. More details on the structure of the generated constraint graph can be found in [12].

B. Schedule Generation

From a generated constraint graph, we can compute a shortest schedule satisfying the graph's time constraints. This involves computing the longest paths from the constraint graph's *zero node*, a node representing an (artificial) event at time 0. These longest paths can be computed if and only if the constraint graph does not have positive cycles. This computation involves (1) multiplying all arc weights in the constraint graph by -1 and (2) computing shortest paths from the zero node using the Bellman-Ford algorithm [13].

A computed schedule for three books is shown in Fig. 12. The first two books, shown in purple and teal in the schedule,

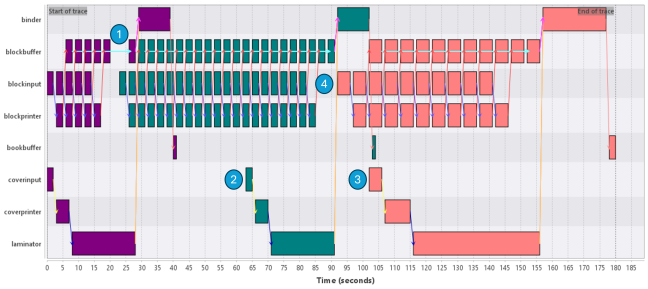

Fig. 12. Generated schedule for three books.

are small books with 5- and 20-sheet book blocks, respectively; the third book, shown in red, is a large book with a 10-sheet book block. The schedule shows that for the first book, the laminator is the bottleneck: the retrieval of the book block is delayed until the lamination is finished (see label 1). For the second and third book, the book block printing is the bottleneck. The handling of the cover sheets is delayed (see labels 2 and 3) to ensure arrival at the book binder together with the corresponding book block. Fig. 12 also shows the setup between small and large books: there is a large delay between retrieval of the last sheet of the second book block and the first sheet of the third book block (see label 4).

C. Performance Analysis

The performance analysis should be quick to make it a feasible approach for production lines in which the workload (and equipment availability) changes dynamically. To assess the methodology's analysis speed, we ran book binding scenarios on a laptop with a 2.7 GHz AMD Ryzen 7 7735U CPU and 32 GB of RAM running Windows 11 24H2. The measurements involve running scenarios from Eclipse IDE for DSL developers 2025-09[2] using Java 21.0.8[3] and data structures and algorithms of jGraphT 1.5.2.[4]

For batches of books with 100-sheet book blocks, Table I shows the batch size, the constraint graph size in terms of nodes (N) and arcs (A), and the schedule synthesis time, i.e., the total time for constraint graph generation and schedule computation. The results show that the constraint graph size scales linearly with the number of books, but the schedule synthesis duration does not. This is caused by the $O(NA)$ complexity of the Bellman-Ford algorithm. Its non-linear complexity comes from the negative-weight arcs, which require re-evaluation of nodes.

Since recently, there are near-linear-time single-source shortest path algorithms for graphs with positive and negative arc weights [14], [15]. We have not applied these to decrease the schedule synthesis duration, because our methodology offers a simpler mechanism to deal with the non-linear scaling of the schedule synthesis duration. A schedule can be created in an incremental manner. The incremental scheduling is based on the *equipment state*, i.e., the last operations

[2]https://www.eclipse.org/
[3]https://www.oracle.com/java/
[4]https://jgrapht.org/

TABLE I

SCHEDULE SYNTHESIS PERFORMANCE.

#books	#nodes (N)	#arcs (A)	synthesis
5	3,061	7,647	0.2 s
10	6,121	15,302	1.0 s
15	9,181	22,957	2.7 s
20	12,241	30,612	5.5 s
25	15,301	38,267	9.4 s
30	18,361	45,922	16.5 s
35	21,421	53,577	23.0 s
40	24,481	61,232	38.0 s
45	27,541	68,887	42.2 s
50	30,601	76,542	55.5 s

performed on all equipment nodes and the corresponding completion times. The earliest start of the operations of a new product to be scheduled only depends on this equipment state, because we assume that there are no time constraints between the operations of the new product and those of earlier products except for equipment availability constraints. Verriet [16] explains how the constraints with respect to equipment state can be captured in a constraint graph.

New products can be added to an existing schedule in small increments taking into account the equipment state. For the book binding case, this means that one can incrementally schedule 10 sets of 5 books, each taking a few tenths of a second instead of scheduling 50 books at once, which takes more than 50 seconds. This way, one obtains linear scaling instead of the non-linear time increase shown in Table I.

To assess the benefits of our methodology for integrally scheduling complete production lines in a realistic setting, we performed experiments with a book printing setup consisting of two machines, a (re-entrant) production printer and a booklet maker. The production printer uses the scheduler of Van Pinxten et al. [3]. In its standard configuration, the printer conservatively delivers sheets to the booklet maker taking into account the booklet maker's longest production time. By integrally scheduling the complete production line of two machines, up to 75% reduction in production time for thin booklets can be achieved [17].

VI. CONCLUSION

In this paper, we present a methodology for scheduling production lines, i.e., fully automated production systems with limited internal buffer capacity. The methodology targets high-mix low-volume applications where the allocation of operations to machines and the ordering of operations on machines is given. The methodology consists of two elements: (1) a domain model to describe a production line and its allocated workload and (2) a transformation to constraint graphs to calculate a shortest feasible schedule of the workload considering all time constraints. The constraint graphs capture the availability of the machines in the production line in terms of operation duration, transport times, and sequence-dependent setup times. We illustrate the methodology on a book binding case from the professional printing domain. Experiments show that online schedule synthesis is feasible. Experiments with a realistic setup show that integrally scheduling a complete production line improves productivity compared to scheduling machines individually.

ACKNOWLEDGMENT

The research is carried out as part of the AIMS program under the responsibility of TNO-ESI in cooperation with Canon Production Printing. The research activities are co-funded by Holland High Tech — TKI HSTM via the PPP Innovation Scheme (PPP-I) for public-private partnerships.

The authors would like to thank Eindhoven University of Technology's Computer Science department for the student collaboration on developing PLOT and Parisa Farboud for performing practical experiments.

REFERENCES

[1] D. C. Ku and G. De Micheli, "Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 696–718, 1992.

[2] U. Waqas, M. Geilen, J. Kandelaars, L. Somers, T. Basten, S. Stuijk, P. Vestjens, and H. Corporaal, "A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 573–578.

[3] J. van Pinxten, U. Waqas, M. Geilen, T. Basten, and L. Somers, "Online scheduling of 2-re-entrant flexible manufacturing systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, 2017.

[4] K. R. Baker and D. Trietsch, *Principles of sequencing and scheduling*, 2nd ed., ser. Wiley Series in Operations Research and Management Science. Wiley, 2019.

[5] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Comput. Oper. Res.*, vol. 142, 2022.

[6] P. Fattahi, M. Saidi Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *J Intell Manuf*, vol. 18, pp. 331–342, 2007.

[7] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Comput. & Oper. Res.*, vol. 73, pp. 165–173, 2016.

[8] B. Naderi, R. Ruiz, and V. Roshanaei, "Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook," *INFORMS J Comput.*, vol. 35, 2023.

[9] B. van der Sanden, Y. Blankenstein, R. Schiffelers, and J. Voeten, "LSAT: Specification and analysis of product logistics in flexible manufacturing systems," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 1–8.

[10] J. Tsai, S. Jennhwa Yang, and Y.-H. Chang, "Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications," *IEEE Trans. Software Eng.*, vol. 21, pp. 32–49, 1995.

[11] J. van Wanrooij, T. Basten, and M. Geilen, "Schedule synthesis for synchronous dataflow models with lower and upper timing bounds," *ACM Trans. Embed. Comput. Syst.*, vol. 24, pp. 18–37, 2025.

[12] J. Verriet, "Production line performance optimisation," TNO, Eindhoven, Tech. Rep. 2025 R11609, 2025. [Online]. Available: https://repository.tno.nl/SingleDoc?docId=74057

[13] R. Bellman, "On a routing problem," *Q. Appl. Math.*, vol. 16, pp. 87–90, 1958.

[14] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen, "Negative-weight single-source shortest paths in near-linear time," *Commun. ACM*, vol. 68, pp. 87–94, 2025.

[15] K. Bringmann, A. Cassis, and N. Fischer, "Negative-weight single-source shortest paths in near-linear time: Now faster!" in *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, Santa Cruz, CA, 2023.

[16] J. Verriet, "Compositional performance prediction for tightly coupled manufacturing systems," TNO, Eindhoven, Tech. Rep. 2023 R12451, 2023. [Online]. Available: https://repository.tno.nl/SingleDoc?docId=56830

[17] P. Farboud, "Context-aware scheduling in production printing," EngD Thesis, Technische Universiteit Eindhoven, 2025. [Online]. Available: https://research.tue.nl/nl/publications/context-aware-scheduling-in-production-printing/